Scalable Inference in Hierarchical Generative Models

Thomas Dean
Department of Computer Science
Brown University, Providence, RI 02912

Abstract

Borrowing insights from computational neuroscience, we present a family of inference algorithms for a class of generative statistical models specifically designed to run on commonly-available distributed-computing hardware. The class of generative models is roughly based on the architecture of the visual cortex and shares some of the same structural and computational characteristics. In addition to describing several variants of the basic algorithm, we present preliminary experimental results demonstrating the pattern-recognition capabilities of our approach and some of the characteristics of the approximations that the algorithms produce.

1 Introduction

Lee and Mumford [9] propose a generative model of the visual cortex that combines top-down and bottom-up inference, employs invariant and compositional representations for encoding and retrieving patterns, and supports associative recall, pattern completion and sequence prediction among other functions. Their model is cast as a graphical statistical model providing clear semantics in terms of a joint distribution over a set of random variables representing features of the target pattern space. While there are many inference algorithms for graphical models, none appear up to the task of learning parameters and absorbing evidence on the scale of the primate visual cortex. We propose to remedy this situation.

Figure 1 shows the first four regions of the temporal cortex — V1, V2, V4 and the inferotemporal cortex (IT) — illustrated, on the left, as a stack of increasingly abstract visual features and their associated processing units, and, on the right, as they are arranged on the cortical sheet. The figure also depicts the postulated interaction between regions implementing both top-down and bottom-up communication. The bottom-up communications are used to combine more primitive features into more abstract ones and the top-down communications are used to exploit expectations generated from prior experience.

Figure 2 derives from Figure 1 by associating each cortical region with a random variable encoding knowledge about the probabilistic relationships among features in a hierarchy of such features. Information propagates up and down the hierarchy and the marginal probabilities associated with the variables are updated to reflect the evidence presented to the bottommost level.

Let x_{V1}, x_{V2}, x_{V4} and x_{IT} represent the four cortical regions and x_{O} the observed data (input). Using the chain rule and the simplifying assumption that in the sequence $(x_{\mathrm{O}}, x_{\mathrm{V1}}, x_{\mathrm{V2}}, x_{\mathrm{V4}}, x_{\mathrm{IT}})$ each variable is independent of the other

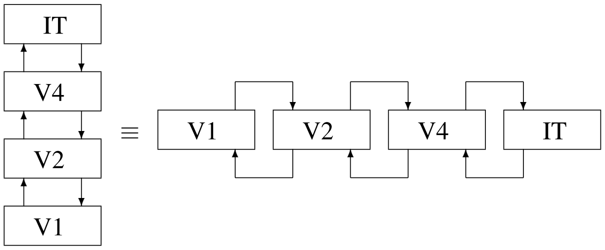


Figure 1: A depiction of the first four regions of the temporal cortex

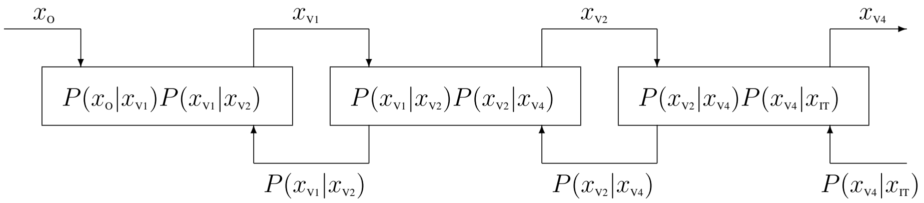

x_{O}
x_{V1}
x_{V2}
x_{V4}
$P(x_{\text{O}}|x_{\text{V1}})P(x_{\text{V1}}|x_{\text{V2}})$
$P(x_{\text{V1}}|x_{\text{V2}})P(x_{\text{V2}}|x_{\text{V4}})$
$P(x_{\text{V2}}|x_{\text{V4}})P(x_{\text{V4}}|x_{\text{IT}})$
$P(x_{\text{V1}}|x_{\text{V2}})$
$P(x_{\text{V2}}|x_{\text{V4}})$
$P(x_{\text{V4}}|x_{\text{IT}})$

Figure 2: The hierarchical Bayesian model proposed by Lee and Mumford. The activity in the ith region is influenced by bottom-up feed-forward data x_{i-1} and top-down probabilistic priors $P(x_i|x_{i+1})$ representing the feedback from region $i+1$.

variables given its immediate neighbors in the sequence, we write the equation relating the four regions as

$$P(x_{\text{O}}, x_{\text{V1}}, x_{\text{V2}}, x_{\text{V4}}, x_{\text{IT}}) = P(x_{\text{O}}|x_{\text{V1}})P(x_{\text{V1}}|x_{\text{V2}})P(x_{\text{V2}}|x_{\text{V4}})P(x_{\text{V4}}|x_{\text{IT}})P(x_{\text{IT}})$$

resulting in a graphical model or *Bayesian network* based on the chain of variables:

$$x_{\text{O}} \leftarrow x_{\text{V1}} \leftarrow x_{\text{V2}} \leftarrow x_{\text{V4}} \leftarrow x_{\text{IT}}$$

The primate cortex consists of approximately 10^{11} sparsely-connected neurons organized in columnar structures each of which is composed of roughly 10^4 cells. Even if we assign only one variable to model each columnar structure we would end up with a network consisting of more than 10^7 variables, a formidable challenge for any inference algorithm.

2 Generalized Pyramid Graphs

Clearly the cortex has a structure that is significantly more complicated than indicated by the simple schematic shown in Figure 1. To model both the hierarchical arrangement of regions and the structure within regions, we employ a class of acyclic graphs called *pyramid graphs*, a term borrowed from [11] in which it is used to characterize a class of Bayesian networks for which *loopy belief propagation* works reasonably well.

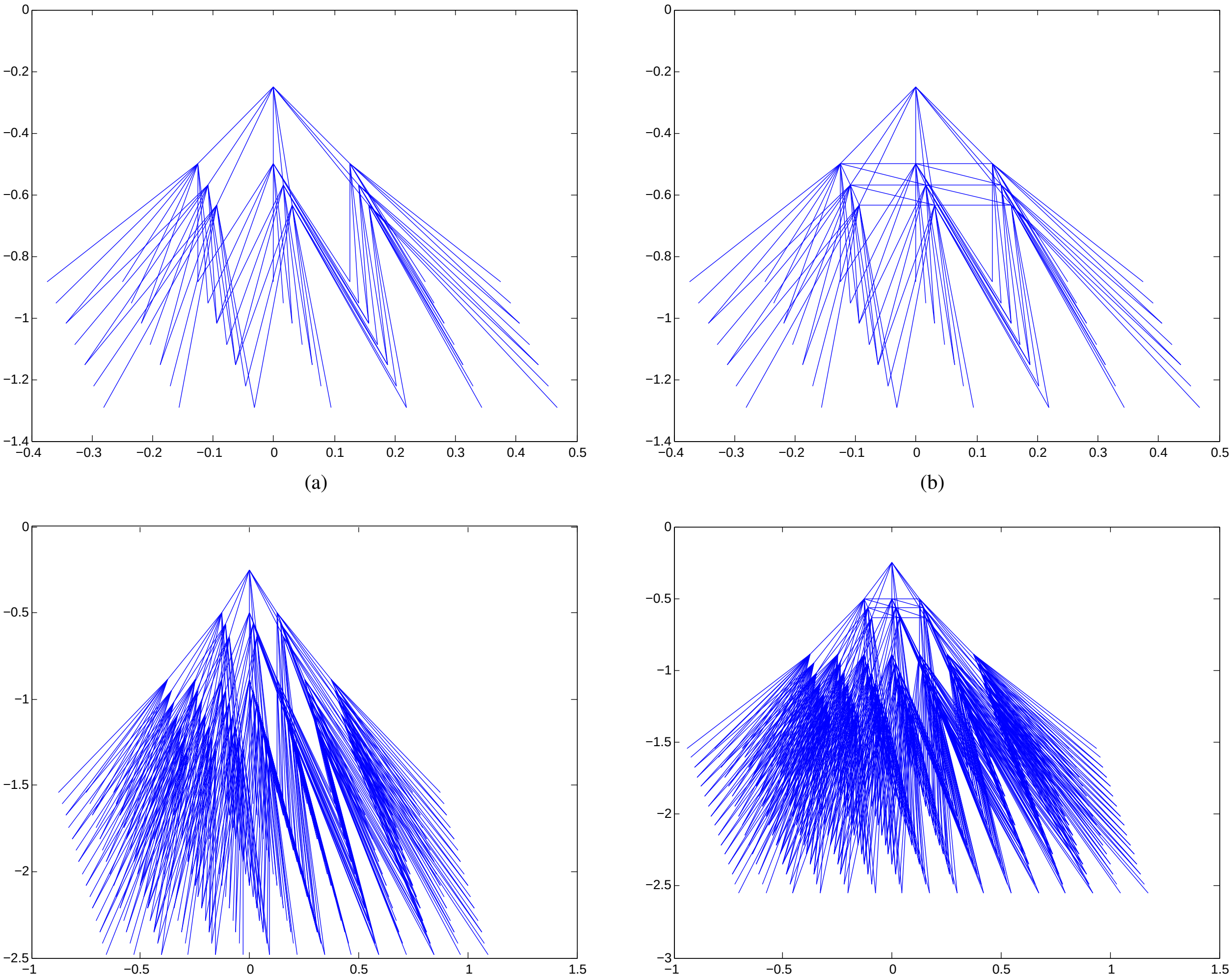
A *generalized pyramid graph* is a directed acyclic graph $G = (V, E)$ whose $N = |V|$ nodes are partitioned into K levels, with level one designating the bottommost or *input* level and level K designating the topmost or *root* level. Each directed edge in E connects a node in level k to a node in level $k-1$ (*inter-level edges*) or to another node in level k (*intra-level edges*). Each *child* level k has a unique *parent* level $k+1$ for all k such that $0 < k < K$. Let i be a node in level k and $f_i = \{j : (i,j) \in E\}$ be the set of children of i in level $k-1$; f_i is said to be the *receptive field* of i. For any two adjacent nodes i and j in level k, their respective receptive fields f_i and f_j may or may not overlap.

The nodes in each level are arranged in a square grid. Two nodes in a given level are *grid adjacent* if they are next to one another in the grid either along a diagonal or along one of the four directions aligned with the grid axes. Two nodes that are grid adjacent may or may not be adjacent in G. The nodes comprising a given receptive field are arranged in a square sub grid of the grid for the level containing the receptive field.

The receptive fields for nodes that are grid adjacent to one another in a given parent level can overlap or share nodes in the child level. It is also possible for the receptive fields for two nodes that are not grid adjacent in a given level to overlap in the child level, for example, in the case in which the overlap for a level is greater than half the width of the receptive fields for that level. The overlap for a given level is specified as an integer used to determine how two receptive fields share nodes depending on the arrangement of the nodes associated with the receptive fields in the parent level. Figure 3 depicts four generalized pyramid graphs that we will discuss at some length in Section 6.

3 Pyramidal Bayesian Networks

A *graphical model* is a compact representation of a joint probability distribution over a set of random variables whose conditional dependencies are completely characterized by a graph in which each random variable is associated with a vertex. Bayesian networks (BNs) and Markov random fields (MRFs) are examples of graphical models. In a given



(a) (b)

(c) (d)

Figure 3: The four pyramid-graph Bayes networks referred to in the text

graphical model, some of the nodes are *observed* and are assigned values subsequent to inference and the remaining variables are *hidden*. A graphical model is *parameterized* by assigning each node in the graph a conditional probability density (CPD) (in the case of directed models such as Bayesian networks) or a potential function (in the case of undirected Markov random fields).

We use generalized pyramid graphs to represent the dependency structure for a class of graphical models called *pyramidal Bayesian networks* (PBNs). Level one corresponds to the input level — the level in which evidence is added to the Bayesian network — and consists of all the terminal nodes in the Bayesian network.

A given level is either *homogeneous*, meaning all the nodes in the level have the same CPDs, or *inhomogeneous*, meaning their parameters differ and are established separately. In the case of homogeneous levels, the parameters for all nodes in a given level are *tied* to expedite inference. We limit the families of allowable densities to just Gaussians and tabular CPDs though the methods described here can be extended to a broader class of continuous densities using non-parametric belief propagation methods [13, 7]. A level can have a mix of discrete (tabular) and continuous (Gaussian) nodes.

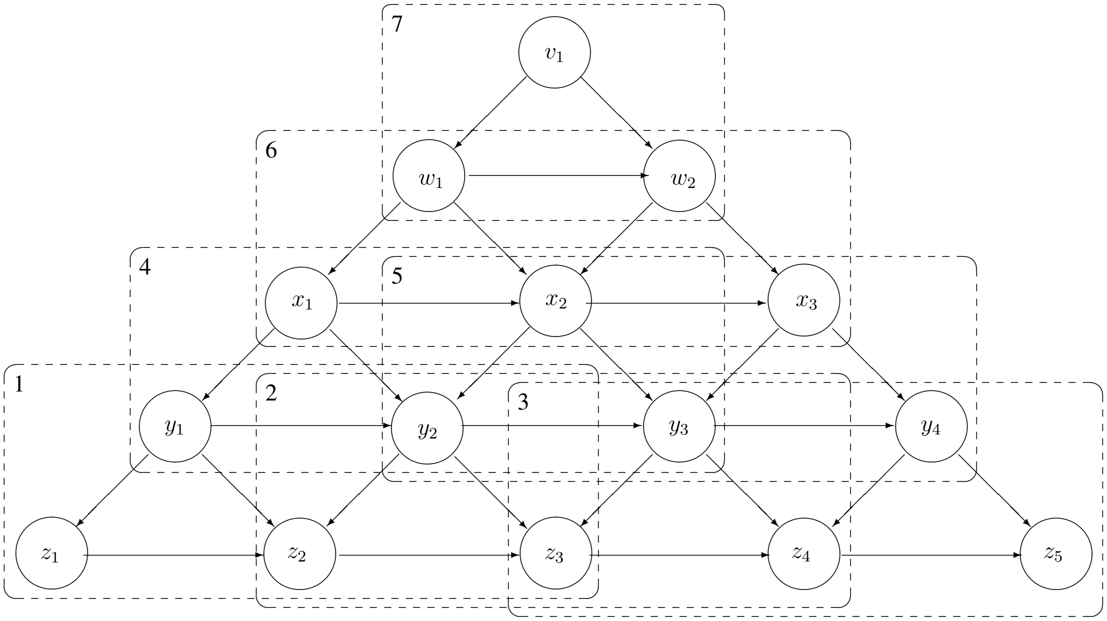


Figure 4: The component problems (subnets) associated with a simple five-level pyramid-graph Bayes network

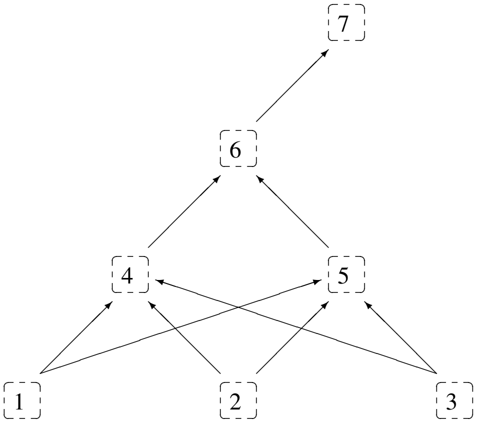


Figure 5: The subnet graph associated with the decomposition of the pyramid-graph Bayes network shown in Figure 4

4 Hierarchical Expectation Refinement

In this section, we describe an algorithm called *hierarchical expectation refinement* that operates by decomposing the problem of learning a large pyramidal Bayesian network (PBN) into a set of more tractable component learning problems. The problem of estimating the parameters of a PBN is decomposed into K subproblems, one for each level. Each of these subproblems is further subdivided into smaller component learning problems corresponding to graph fragments that are used to create small Bayesian networks — referred to here as *subnets* — that can be solved in isolation. Learning proceeds level-by-level from the bottom up starting from the input level. Learning within a level can proceed simultaneously, in parallel on all the subnets associated with that level. For simplicity of presentation, we assume that all the nodes in the BN are discrete and their corresponding CPDs are represented as conditional probability tables (CPTs). We begin with the example shown in Figure 5.

Figure 4 depicts a five-level PBN showing the decomposition of the first layer into three subnets. The three dashed boxes enclose the variables involved in each of the component learning problems associated with the first layer. All edges that extend outside of a dashed box are eliminated for purposes of parameter estimation, thereby isolating a relatively small and thus tractable subgraph characterized by the following joint probability distribution for the learning component labeled 1 in Figure 4.

$$P(z_1, z_2, z_3, y_1, y_2) = P(z_1|y_1)P(z_2|z_1, y_1, y_2)P(z_3|z_2, y_2)P(y_2|y_1)P(y_1)$$

We estimate the parameters for this subnet using expectation maximization with a training set of assignments to the variables $\{z_1, z_2, z_3\}$ — the set of variables $\{y_1, y_2\}$ are hidden. We use the estimated parameters to quantify $P(z_1|y_1)$ and $P(z_2|z_1, y_1, y_2)$ in the PBN and throw away the rest.

Generalizing, suppose we want to learn the parameters for a node in level k having already learned the parameters for all nodes in levels $k-1, k-2, \ldots, 1$. Recall that for any node in level k, its parents are either in level k or $k+1$. We define the *extended family* of a node x in level k as the parents of x and the children of those parents of x that are in level k. To learn the parameters associated with a node x in level k (the parameters for $P(x|\texttt{Parents}(x))$), we construct a component BN with a set nodes corresponding to the extended family of x. In the component BN, the nodes in level k are observed and those in level $k + 1$ are hidden. We initialize the CPTs of variables in the component BN to random distributions with Dirichlet priors, collect a sufficiently large set of samples of the hidden nodes in the component, and use this set of samples as a training set for learning the component parameters. We assign the parameters of the CPT for x in the PBN to be the same as those of the CPT for x in the component BN.

In the most straightforward version of the learning algorithm, we learn all the parameters for variables in level k before proceeding to level $k + 1$. In learning the parameters for variables in level one, the training set of assignments to variables in level one correspond to inputs. In the case of the assignments required for estimating the parameters of variables at level $k > 1$, the training data is acquired by sampling from the posterior *belief* function $\texttt{Bel}(\vec{x}) = P(\vec{x}|\xi)$ where $\vec{x}$ is the vector of (observed) variables in the component learning problem at level k and ξ is an assignment to the (observed) variables in the input level of the PBN.

In the hand-written-digit-recognition experiments described in Section 6, we learn the parameters for all but the root and penultimate levels in an unsupervised manner. The subnets corresponding to the penultimate level (typically there is only one) are then trained as a classifier in a supervised manner using the supplied training labels.

There are a number of obvious optimizations. Instead of creating a separate component problem for each node in a level, we compute the maximal sets [15] of the collection of sets corresponding to the extended families for each node. We then use these maximal sets to construct the set of component BNs. When we estimate the parameters of a component BN, we use the learned parameters to quantify all those variables whose extended family is a subset of the maximal set associated with the component BN, thereby reducing the total number of component problems considerably. For a homogeneous level, we construct a single component BN which effectively ties the parameters for all the nodes in that level.

We use several algorithms for inference. Exact inference on general graphical models is NP hard. Even so, fast algorithms exist that perform exact inference on graphical models consisting of several hundreds or even thousands of nodes. For exact inference, we use an implementation of Lauritzen and Spiegelhalter's [8] junction-tree algorithm (referred to as JTREE in the following).

For approximate inference, we use loopy belief propagation [11] implemented using one of two belief-propagation algorithms: Pearl's original message-passing algorithm [12] (PEARL) and a variant (BELPROP) based on Aji and McEliece's GDL (Generalized Distributive Law) algorithm that exhibits better convergence [1] on our networks. As an alternative to JTREE, PEARL and BELPROP, we can also exploit the structure induced by the component BNs to perform approximate inference.

Figure 5 depicts a graph $G = (V, E)$ — the *subnet graph* — in which the vertices are subnets $V = \{s_i\}$ and there is an edge $(s_i, s_j) \in E$ just in case s_i has a hidden node which is an observed node of s_j. We can use this subnet graph to perform inference by propagating samples from the input level to the root as follows: As soon as a subnet's observed nodes are instantiated, we compute the subnet's posterior distribution and the MAP assignment to its hidden variables. This assignment is then propagated to the subnet's parent subnets in the subnet graph. We refer to this algorithm as SUBNET in the following.

5 Implementation and Evaluation

The approach described in the previous section was designed to be implemented on a compute cluster using MPI (Message Passing Interface) code to handle communication between processes responsible for the component learning problems. While we haven't yet realized a parallel implementation, we have developed a non-parallel prototype in Matlab using Kevin Murphy's Bayes Net Toolbox [10] which is available for download at `http://www.cs.brown.`

edu/~tld/projects/cortex/prototype.tar.gz and includes all of the code required to replicate the experiments discussed in this section.

We tested our ideas on the problem of recognizing handwritten digits. Yann LeCun has taken a subset of images from the National Institute of Science and Technology's (NIST) database of handwritten digits, size-normalized each digit, and centered it in a fixed-sized image to produce a data set consisting of 30,000 training images of digits produced by a set of 250 writers and 10,000 test images produced by an additional set of 250 writers disjoint from the first set. LeCun's data set is available at http://yann.lecun.com/exdb/mnist/.

In our experiments on the NIST hand-written-digit data set, we consider four PBNs, the first of which is small enough that we can perform all of the required inference directly on the global model (the full PBN) and compare the performance on the global model with the performance on various decompositions of the global model. Inference plays three different roles: (1) inference is required to generate the samples required for local-model (subnet) parameter estimation, (2) once a sufficient number of samples has been obtained, inference is required to estimate the parameters of the local models, and (3) once the model parameters have all been learned, inference is required for evaluation purposes. If the global model is to be used for inference, then the local-model parameters are used to update the parameters of the global model.

To investigate the factors governing performance, we use different algorithms in each of the three roles where inference is required. For instance, we might generate samples and perform evaluation using an exact inference algorithm applied to the global model, but update the parameters of the global model from the local models. Such an experiment provides insight into how well the local models perform in learning generative models. We can then substitute an approximate inference algorithm that is more likely to scale with the size of the global model and evaluate the impact of the approximation on performance.

6 Preliminary Experimental Results

The first PBN was designed with two characteristics in mind: (1) create a network that exhibits the features we consider most important in this architecture: multiple levels, overlapping receptive fields, and intra-level edges, and (2) make the network small enough that it is possible to perform exact inference on the global model. This first PBN is shown in Figure 3.a and summarized as follows (WIDTH, OVERLAP and INTRA refer to, respectively, the width of the receptive fields, the number of rows/columns of overlap, and whether or not there are intra-level edges with respect to a given level):

LEVEL	WIDTH	OVERLAP	INTRA	SIZE	SUBNETS
4	1	0	0	1×1	1
3	3	1	1	3×3	1
2	3	1	0	7×7	4
1	4	0	0	28×28	NA
TOTAL				843	6

The input level maps directly onto the 28×28 eight-bit images of digits comprising the data set. The largest subnet has 29 nodes. The total number of parameters is 1,058,064. In this and all the models that follow, the maximum number of parameters in a single CPD is 150,000, corresponding to a random variable with 16 possible states and four parents each of which has 10 possible states. In all of the PBNs, the input level is homogeneous consisting of conditional Gaussian nodes implementing a mixture-of-Gaussians classifier applied to either a 3×3 or 4×4 image patch. All the other levels are inhomogeneous and all the other nodes are discrete.

In describing each experiment, we specify the number of patches used to train the input level (# PATCH), the number of images used for training the other levels (# TRAIN), the number of images used to evaluate performance (# TEST), the percentage of training images correctly classified (% TRAIN), the percentage of testing images correctly classified (% TEST), and the CPU hours required for training and evaluation (# HOUR). In addition, we identify the inference algorithm used for each of the three roles mentioned above by specifying a triple, $\mathrm{ALG}_1/\mathrm{ALG}_2/\mathrm{ALG}_3$, where ALG_i is one of JTREE, PEARL, BELPROP, SUBNET or NA, and ALG^2 and ALG^3 serve as shorthand for ALG/ALG and ALG/ALG/ALG respectively; JTREE/NA/JTREE serves as a benchmark for the smaller networks. Here are the results of some representative experiments that emphasize the implications of using SUBNET to estimate the parameters of the global model:

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
SUBNET/JTREE^2	10,000	1,000	1,000	99.50	52.70	2.89
SUBNET/JTREE^2	20,000	10,000	5,000	95.90	78.86	20.16
SUBNET/JTREE^2	40,000	20,000	10,000	93.24	81.34	74.55
JTREE/NA/JTREE	10,000	1,000	1,000	100.00	50.30	2.18
JTREE/NA/JTREE	20,000	10,000	5,000	99.45	81.34	19.85

If we substitute loopy belief propagation for exact inference, we get mixed results due to the fact that PEARL failed to converge on most of the training and testing examples (BELPROP also occasionally failed to converge but on a much smaller set):

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
SUBNET/PEARL^2	10,000	400	200	46.25	19.50	9.97
SUBNET/BELPROP^2	10,000	1,000	1,000	75.20	45.70	44.67
BELPROP/NA/BELPROP	10,000	1,000	1,000	88.20	47.60	29.62

If we use SUBNET for sampling and evaluation, we observe a pattern in which performance on the training set falls off if we use SUBNET just for sampling and JTREE for evaluation, but is restored if we use SUBNET in both roles:

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
SUBNET^2/JTREE	10,000	1,000	1,000	69.20	40.90	2.57
SUBNET^2/JTREE	20,000	10,000	5,000	65.22	54.54	20.17
SUBNET^2/JTREE	40,000	20,000	10,000	70.83	69.06	42.77
SUBNET^3	10,000	1,000	1,000	99.90	38.10	3.45
SUBNET^3	20,000	10,000	5,000	97.30	66.98	17.45
SUBNET^3	40,000	20,000	10,000	95.71	77.69	39.17

The PBN shown in Figure 3.b is the same as that shown in Figure 3.a except we eliminate the intra-level edges in level 3 in order to evaluate the impact of intra-level edges. This PBN also has far fewer parameters than our first experimental model, 658,572 versus 1,058,064.

LEVEL	WIDTH	OVERLAP	INTRA	SIZE	SUBNETS
4	1	0	0	1×1	1
3	3	1	0	3×3	1
2	3	1	0	7×7	4
1	4	0	0	28×28	NA
TOTAL				843	6

We get some reduction in performance when using exact inference for sampling and evaluation:

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
SUBNET/JTREE^2	10,000	1,000	1,000	78.00	62.50	2.89
SUBNET/JTREE^2	20,000	10,000	5,000	74.03	69.68	20.16
JTREE/NA/JTREE	10,000	1,000	1,000	97.00	70.08	6.19
JTREE/NA/JTREE	20,000	10,000	5,000	92.43	82.40	49.03

but PEARL and BELPROP converge in all cases. The running times for PEARL and BELPROP have more to do with Matlab's relatively poor performance on problems that cannot be easily vectorized than they have to do with the algorithms themselves which are desirable because they can be implemented to exploit parallelism.

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
SUBNET/PEARL^2	10,000	1,000	1,000	73.30	63.40	60.08
SUBNET/BELPROP^2	10,000	1,000	1,000	76.20	63.20	33.55
SUBNET^3	10,000	1,000	1,000	81.50	67.70	1.81
SUBNET^3	20,000	10,000	5,000	78.33	75.24	17.68
BELPROP/NA/BELPROP	10,000	1,000	1,000	83.10	63.70	47.10

The PBN shown in Figure 3.c was designed to test the impact of adding an additional level. The resulting network has 42 subnets, the largest of which has 40 nodes. The total number of parameters is 23,346,720:

LEVEL	WIDTH	OVERLAP	INTRA	SIZE	SUBNETS
5	1	0	0	1×1	1
4	3	1	1	3×3	1
3	3	1	0	7×7	4
2	4	2	0	16×16	36
1	4	2	0	34×34	NA
TOTAL				1471	42

This network was too large for the other algorithms and performed less well than the previous two but still reasonable when one considers the ratio of of training examples to total number of parameters. Further experiments on networks of this size will have to wait for the parallel implementation, which should reduce the training and evaluation time to less than an hour for the case of 1,000 training and testing examples.

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
SUBNET^3	10,000	100	100	100.00	30.00	6.27
SUBNET^3	10,000	1,000	1,000	100.00	31.20	43.05

Finally, the PBN shown in Figure 5.d was designed to reduce the total number of parameters somewhat while using smaller receptive fields and hence encouraging more localized features in the lower levels. Figure 5.d uses 3×3 receptive fields with an overlap of two rows/columns at the lowest level. It has the same number of subnets as the previous network but only 6,315,228 parameters total:

LEVEL	WIDTH	OVERLAP	INTRA	SIZE	SUBNETS
5	1	0	0	1×1	1
4	3	1	1	3×3	1
3	3	1	0	7×7	4
2	3	2	0	15×15	36
1	3	2	0	31×31	NA
TOTAL				1245	42

Performance is respectable on the few experiments run so far with some reduction in performance on the training data, but good performance on the test examples. As in other experiments involving SUBNET^3, performance on the training data goes down slightly (down 3%) while testing performance improves (up 10%).

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
SUBNET^3	10,000	1,000	1,000	83.00	65.00	12.79
SUBNET^3	20,000	10,000	5,000	80.00	75.00	191.96

SUBNET^3 performs inference in a purely bottom-up fashion unlike other algorithms such as SUBNET/PEARL^2 that propagate information both up and down the hierarchy. One obvious alternative is to perform a variant of belief propagation that operates directly on the subnet graph. Subnets can then serve as they do in other algorithmic variants to couple nearby features and thus serve both diagnostic and causal inference.

To test this idea, we implemented an instance of the *generalized belief propagation* framework described in [14] which is referred to as GBP in the following. In GBP, belief propagation is carefully orchestrated in three passes of the subnet graph. At the start of the first pass, subnets on the input level incorporate the new evidence and propagate marginal distributions over shared variables to level-two subnets adjacent in the subnet graph. The marginals are computed using the posterior distribution (which now incorporates the new evidence) to marginalize out variables in the intersection of the two adjacent subnets.

Evidence continues to propagate up the subnet graph until it reaches the highest level in which all of the subnet parameters have already been learned. In the second pass, marginals are propagated back down to the input level. The third pass is essentially the same as the first pass and completes the process of distributing the new evidence throughout the subnet graph.

To get some idea of how the additional passes impact performance we compare GBP to the SUBNET^3 algorithm and our benchmark JTREE/NA/JTREE. The next table shows the results for a set of experiments on the network shown in Figure 3.a. We note that GBP^3 significantly outperforms SUBNET^3 and compares favorably with JTREE/NA/JTREE.

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
GBP^3	10,000	1,000	1,000	99.00	55.50	3.20
GBP^3	20,000	10,000	5,000	94.79	78.96	29.11
SUBNET^3	10,000	1,000	1,000	99.90	38.10	3.45
SUBNET^3	20,000	10,000	5,000	97.30	66.98	17.45
JTREE/NA/JTREE	10,000	1,000	1,000	100.00	50.30	2.18
JTREE/NA/JTREE	20,000	10,000	5,000	99.45	81.34	19.85

Next we provide the results of a set of experiments on the same network but without the intra-level edges as shown in Figure 3.b. In this case, the difference in performance between GBP^3 and SUBNET^3 is not significant and both manifest a marked reduction in accuracy on the training data when compared with JTREE/NA/JTREE.

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
GBP^3	10,000	1,000	1,000	79.80	68.00	2.75
GBP^3	20,000	10,000	5,000	79.97	75.04	25.81
SUBNET^3	10,000	1,000	1,000	81.50	67.70	1.81
SUBNET^3	20,000	10,000	5,000	78.33	75.24	17.68
JTREE/NA/JTREE	10,000	1,000	1,000	97.00	70.08	6.19
JTREE/NA/JTREE	20,000	10,000	5,000	92.43	82.40	49.03

Finally, we compare the performance of GBP^3 and SUBNET^3 on the network shown in Figure 6.d. We don't as yet have an explanation for why GBP^3 performs less well than SUBNET^3 on the training data but outperforms SUBNET^3 on the testing data, but this behavior was observed in several similar experiments involving this network.

ALGORITHM	# PATCH	# TRAIN	# TEST	% TRAIN	% TEST	# HOUR
GBP^3	10,000	1,000	1,000	77.90	68.70	20.90
SUBNET^3	10,000	1,000	1,000	83.00	65.00	12.79

7 Related Work and Conclusions

We are essentially implementing expectation maximization [2] in a distributed but highly structured model (hierarchically arranged in levels with topologies that embed in the plane — or nearly so). We expect that for many perceptual learning problems the particular way in which we decompose the parameter-estimation problem (which is related to Hinton's weight-sharing and parameter-tying techniques) will avoid potential problems due to early (and permanently) assigning parameters to the lower layers.

We were originally attracted to the idea of level-by-level parameter estimation in reading George and Hawkins' "A hierarchical Bayesian model of invariant pattern recognition in the Visual Cortex" [3]. In the algorithm described in [3], each node in a Bayesian network collects instantiations of its children that it uses to derive the state space for its associated random variable and estimate the parameters for its children's CPDs. The states that comprise the state space for a given node correspond to perturbed versions (exemplars) of the most frequently appearing vectors of instantiations of its children. Their network is singly connected and, for this class of graphs, Pearl's belief-propagation algorithm is both exact and efficient (it scales linearly in the diameter of the underlying graph).

Hinton, Osindero and Teh [6] present a hybrid model combining Boltzmann machines and directed acyclic Bayesian networks and an improvement and generalization on Hinton's method of contrastive divergence [4]. The authors present a greedy, level-by-level approach to learning a generative model similar to the one proposed by Lee and Mumford [9].

SUBNET^3 is medium-grain parallelizable with each subnet running on a separate processor (or, as is often the case, if there are fewer processors than subnets, using the strategy depicted in Figure 6) and communication fixed through the sharing of hidden/observed random variables as specified in the subnet graph. That the local computations yield a

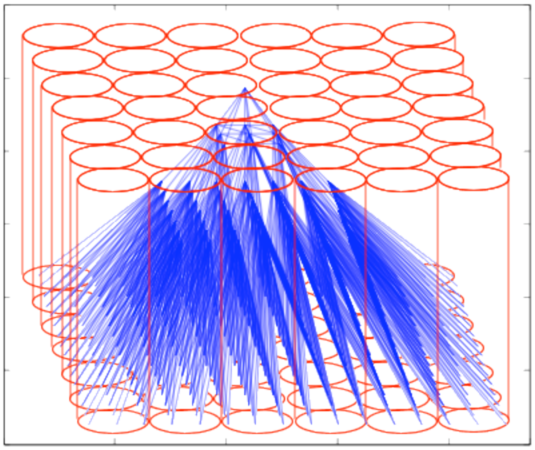


Figure 6: In the parallel implementation, subnets are allocated to processors using a roughly columnar arrangement so that subnets that are adjacent in the subnet graph tend to run on the same processor.

reasonably good approximation of the global joint distribution is very promising. Our preliminary results suggest that inference via intra-level edges is potentially problematic, but we are developing a hybrid model whereby inter-level edges are directed and intra-level edges are undirected with inference performed in a semi-synchronous manner. In this model, which is related to the model described in Hinton, Osindero and Bao [5], propagation is carried out within all levels simultaneously, followed by propagation between all adjacent levels simultaneously, and then repeating this cycle until quiescence.

References

[1] Srinivas M. Aji and Robert J. McEliece. The generalized distributive law. *IEEE Transactions on Information Theory*, 46(2):325–343, 2000.

[2] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society*, 39 (Series B):1–38, 1977.

[3] Dileep George and Jeff Hawkins. A hierarchical Bayesian model of invariant pattern recognition in the visual cortex. In *Proceedings of the International Joint Conference on Neural Networks*. IEEE, 2005.

[4] G. Hinton. Training products of experts by minimizing contrastive divergence. *Neural Computation*, 14:1771–1800, 2002.

[5] Geoffrey E. Hinton, Simon Osindero, and Kejie Bao. Learning causally linked Markov random fields. In Robert G. Cowell and Zoubin Ghahramani, editors, *Proceedings of the 10th International Workshop on Artificial Intelligence and Statistics, January 6–8, 2005, Savannah Hotel, Barbados*, pages 128–135. Society for Artificial Intelligence and Statistics, 2005.

[6] Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Submitted to Neural Computation*, 2005.

[7] M. Isard. PAMPAS: real-valued graphical models for computer vision. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition: Volume I*, pages 613–620, 2003.

[8] Steffen L. Lauritzen and David J. Spiegelhalter. Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society*, 50(2):157–194, 1988.

[9] Tai Sing Lee and David Mumford. Hierarchical Bayesian inference in the visual cortex. *Journal of the Optical Society of America*, 2(7):1434–1448, July 2003.

[10] Kevin Murphy. The Bayes net toolbox for Matlab. *Computing Science and Statistics*, 33, 2001.

[11] Kevin Murphy, Yair Weiss, and Michael Jordan. Loopy-belief propagation for approximate inference: An empirical study. In *Proceedings of the 16th Conference on Uncertainty in Artificial Intelligence*, pages 467–475. Morgan Kaufmann, 2000.

[12] Judea Pearl. Distributed revision of composite beliefs. *Artificial Intelligence*, 33:173–216, 1987.

[13] E. Sudderth, A. Ihler, W. Freeman, and A. Willsky. Nonparametric belief propagation. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition: Volume I*, pages 605–612, 2003.

[14] Jonathan S. Yedidia, William T. Freeman, and Yair Weiss. Understanding belief propagation and its generalizations. In *Exploring artificial intelligence in the new millennium*, pages 239–269. Morgan Kaufmann Publishers Inc., San Francisco, 2003.

[15] Daniel Yellin. Algorithms for subset testing and finding maximal sets. In *Proceedings of the 3rd Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 386–392, 1992.